System Design Week 4

Design A URL Shortener | System Design

What is a URL shortener?

- It has two main functions:
    - Generate a short URL from a long URL.
    - Retrieve the long URL from a short URL.

How to generate a short URL:

1. Use a deterministic hashing function on the long URL.
2. Take the first few characters of the hash as the short URL.
3. Check if the short URL already exists in a database.
    - If it does, modify the long URL slightly and try again (to avoid collisions).
4. Store the mapping between the long URL and the short URL in a database.

Database:

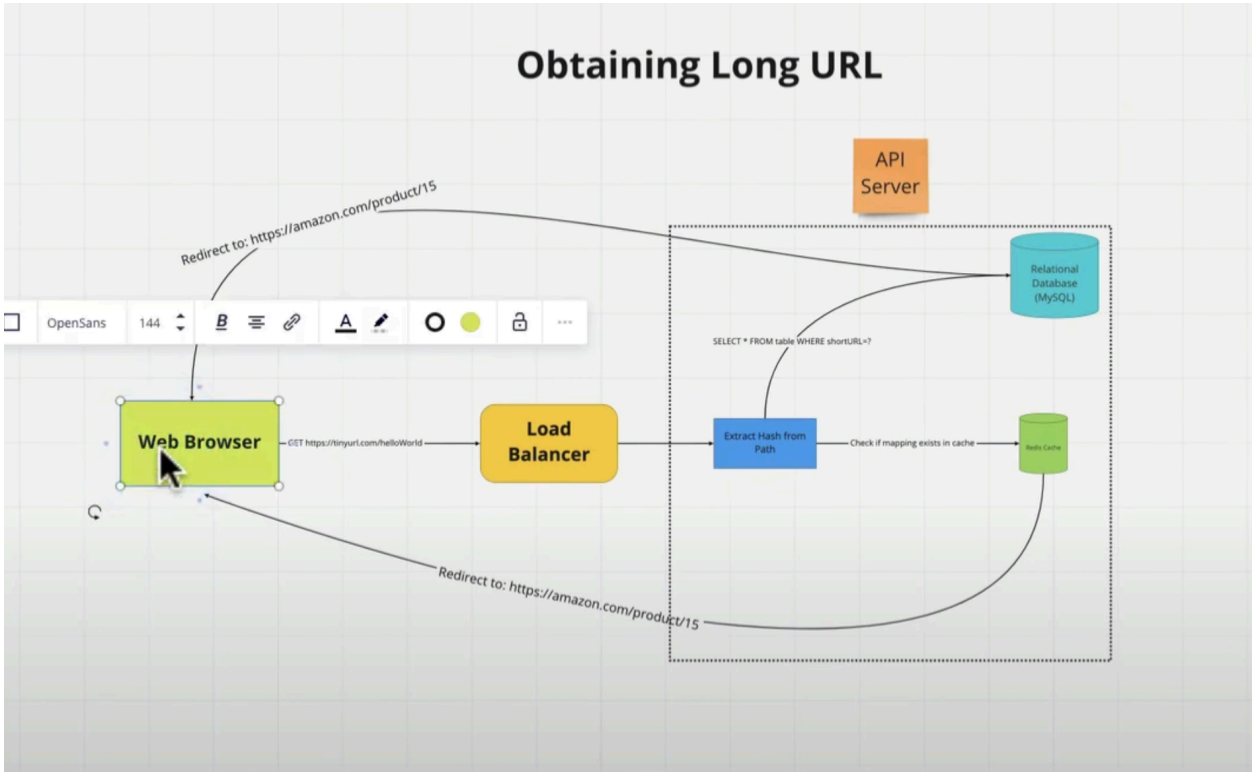
- A table with three columns:
    - ID (auto-incrementing)
    - Long URL
    - Short URL

How to retrieve the long URL:

1. The user enters the short URL in their browser.
2. The browser sends a GET request to the short URL.
3. The server extracts the hash from the short URL.
4. Check the cache (e.g., Redis) for the mapping.
    - If found in the cache, return the long URL.
5. If not found in the cache, check the database for the mapping.
    - If found in the database, return the long URL and update the cache.
    - If not found, return an error.

Performance Improvement:

- Use a cache (e.g., Redis) to store frequently accessed mappings for faster retrieval.



URL Shortening System Design

Functional Requirements (FRs):

- Generate Short URL: Create a short URL for any given long URL.
- Redirect: Redirect a user from the short URL to the corresponding long URL.

Non-Functional Requirements (NFRs):

- High Availability: The service should be available most of the time.
- Low Latency: The service should have minimal response time.

Short URL Length

- The length of the short URL depends on the number of unique URLs the system handles.
- Steps to Calculate Required Length:
- Estimate the number of requests per second (X).
- Multiply X by time factors (seconds, minutes, hours, days, years) to get the total number of requests (Y) the system must support.

- Define the character set for the short URL (e.g., alphabets, numbers).
- Calculate the base of the number system required to represent Y using the character set size (n = log(character set size) (Y)).
- Choose a short URL length corresponding to n (e.g., n = 5 corresponds to a length of 6).

System Architecture

The system architecture consists of the following components:

- UI: Takes the long URL input and shows the short URL output.
- Short URL Service: Generates and stores short URLs in a database; redirects users from short URLs to long URLs.
- Token Service (optional): Manages unique numbers for short URL generation to prevent collisions (avoiding a single point of failure with Redis).

Short URL Generation with Token Service

- The **Short URL Service** requests a token range from the **Token Service**.
- **Token Service** assigns a range of unique numbers (e.g., 1001-2000) to the Short URL Service.
- The Short URL Service converts these numbers to base 62 to create short URLs and stores the mapping in a database like **Cassandra**.
- Once the token range is exhausted, the Short URL Service requests a new range.

Analytics

- The system can collect analytics data about shortened URLs, including:
    - **Source Platform** (e.g., Facebook, LinkedIn)
    - **User Agent** (e.g., browser, mobile app)
    - **Source IP Address** (for geolocation)
- This data can be sent to a **message queue** (e.g., Kafka) for asynchronous processing.
- Uses of analytics include understanding user behavior, making data-driven decisions about data center placement, etc.

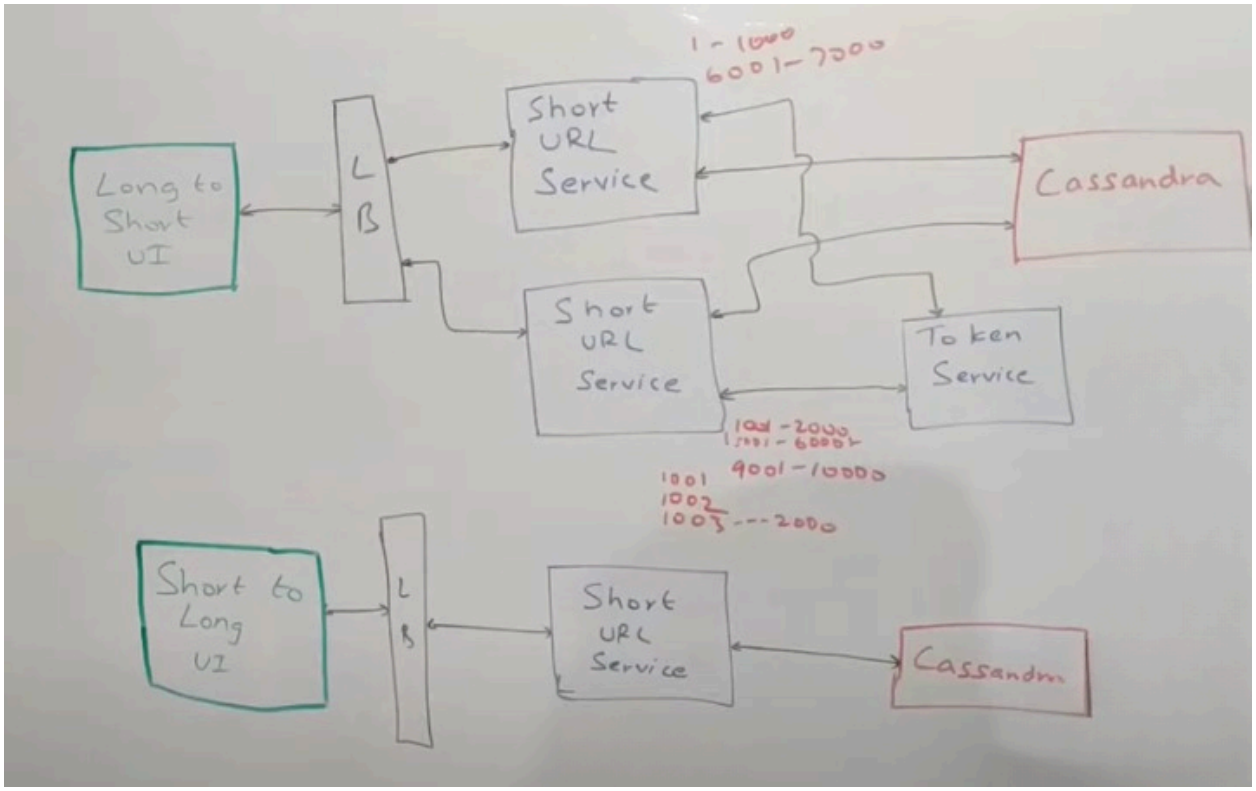
Optimizing Analytics

To reduce the impact on latency when sending data to Kafka:

- **Parallel Calls**: Send data in a separate thread while responding to the user.
- **Batch Writes**: Aggregate data locally and send it periodically or when a threshold is reached.

Trade-offs

- **Asynchronous Writes**: This may lead to some data loss, but might be acceptable for simple analytics.
- **Batch Writes**: Improves performance but increases the risk of larger data loss if a write fails.



Chat Application System Design

Core Components

- **WebSockets:** Used for real-time communication between clients and servers.
- **Chat Service:** Handles message delivery, persistence, and online/offline status.
- **Token Service:** Generates unique numbers for short URLs.
- **Notification Service:** Sends notifications to offline users.
- **Mapping Service:** Maps users to specific WebSocket servers.
- **Databases:** Stores user data, messages, and group memberships.

Data Flow

1. **The user sends a message:**
    - WebSocket server receives the message.
    - The message is written to the database.
    - Mapping Service retrieves the recipient's WebSocket server.
    - The message is forwarded to the recipient's WebSocket server.
2. **The recipient receives a message:**
    - If online, the message is pushed to the user's device.
    - If offline, the message is stored in a queue for later delivery.
3. **Offline user comes online:**
    - A history API is called to retrieve undelivered messages.
    - Messages are pushed to the user's device.

Data Modeling

- **User Table:** Stores user information (ID, password, status, profile image).
- **Group Table:** Stores group information (ID, members).
- **Message Table:** Stores messages (sender, recipient, timestamp, content).
- **User Membership Table:** Maps users to groups.

Key Concepts

- **Partitioning:** Distributing data across multiple nodes for scalability.
- **Clustering:** Sorting data within partitions for efficient retrieval.
- **LSM Trees:** Data structures optimized for write-heavy workloads.
- **NoSQL Databases:** Suitable for handling large amounts of unstructured data.
- **WebSocket:** Protocol for real-time communication.
- **HTTP:** Protocol for client-initiated requests (e.g., joining/leaving groups).
- **Message queues:** Used for asynchronous message delivery.

Additional Considerations

- **Scalability:** Design the system to handle increasing numbers of users and messages.

- **Performance:** Optimize database queries, use caching, and consider load balancing.
- **Reliability:** Implement fault tolerance and redundancy to ensure high availability.
- **Security:** Protect user data and prevent unauthorized access.
- **Testing:** Thoroughly test the system to identify and fix issues.

Detailed Components

- **WebSocket Server:**
    - Handles WebSocket connections from clients.
    - Receives and processes messages.
    - Sends messages to recipients.
    - Manages user sessions.
- **Chat Service:**
    - Stores and retrieves messages from the database.
    - Handles message delivery and retries for offline users.
    - Manages group membership and online/offline status.
- **Token Service:**
    - Generates unique tokens for user sessions.
    - Assign tokens to WebSocket servers.
    - Manages token expiration and renewal.
- **Notification Service:**
    - Sends notifications to offline users via push notifications or email.
    - Stores notification history.
- **Mapping Service:**
    - Maps users to specific WebSocket servers.
    - Handles server load balancing and failover.
- **Databases:**
    - User Database: Stores user information (ID, password, status, profile image).
    - Message Database: Stores messages (sender, recipient, timestamp, content).
    - Group Database: Stores group information (ID, members).
    - User Membership Database: Maps users to groups.

Communication Protocols

- **WebSockets:** Used for real-time communication between clients and servers.
- **HTTP:** Used for client-initiated requests (e.g., joining/leaving groups, retrieving message history).
- **Message Queues:** Used for asynchronous message delivery between services.

Security Considerations

- **Authentication and Authorization:** Implement mechanisms to verify user identity and control access to resources.
- **Data Encryption:** Encrypt sensitive data (e.g., messages, user information) at rest and in transit.

- **Input Validation:** Validate user input to prevent injection attacks and other security vulnerabilities.
- **Rate Limiting:** Limit the number of requests a user can make to prevent abuse.

Testing

- **Unit Testing:** Test individual components and functions.
- **Integration Testing:** Test the interaction between different components.
- **End-to-End Testing:** Test the entire system from the user's perspective.
- **Performance Testing:** Evaluate the system's performance under load.
- **Security Testing:** Identify and address security vulnerabilities.

By carefully considering these factors, you can design a robust and scalable chat application that meets the needs of your users.

System Design: WhatsApp

Key Features

- **One-to-one messaging:** Users can send messages directly to each other.
- **Group messaging:** Users can create and participate in groups.
- **Image/video sharing:** Users can send and receive media files.
- **Read receipts:** Indicate when messages have been read.
- **Online/offline status:** Shows whether users are currently active.

System Architecture

1. **Clients:** User devices (phones, computers) that connect to WhatsApp.
2. **Gateway:** Acts as an intermediary between clients and the backend services.
3. **Session Service:** Manages user sessions, maps users to WebSocket servers, and stores online/offline status.
4. **Message Service:** Handles message delivery, persistence, and routing.
5. **Group Service:** Manages group membership and information.
6. **Notification Service:** Sends notifications to offline users.
7. **Database:** Stores user data, messages, and group information.

Data Flow

1. **User A sends a message to User B:**
    - Client A sends the message to the gateway.
    - Gateway sends the message to the session service.
    - Session service retrieves User B's WebSocket server.
    - Gateway sends the message to User B's WebSocket server.

2. **User B receives the message:**
    - If online, the message is displayed immediately.
    - If offline, the message is stored for later delivery.
3. **Read receipts:**
    - When User B reads the message, they send a read receipt to the server.
    - The server updates the message status and sends a read receipt to User A.

Key Components

- **WebSockets:** Used for real-time communication between clients and servers.
- **Message Queues:** Used for asynchronous message delivery.
- **Databases:** Store user data, messages, and group information.
- **Consistent Hashing:** Used to distribute data across servers efficiently.

Additional Considerations

- **Scalability:** Design the system to handle a large number of users and messages.
- **Performance:** Optimize database queries, use caching, and avoid unnecessary network traffic.
- **Reliability:** Implement fault tolerance and redundancy to ensure high availability.
- **Security:** Protect user data and prevent unauthorized access.
- **Privacy:** Ensure that user messages are encrypted and only accessible to the sender and recipient.

Challenges and Solutions

- **Message delivery:** Ensure reliable delivery, even when users are offline or experiencing network issues. Use message queues and retries.
- **Scalability:** Handle a large number of users and messages. Distribute data across multiple servers and use sharding techniques.
- **Performance:** Optimize database queries, use caching, and minimize network traffic.
- **Security:** Protect user data from unauthorized access and ensure end-to-end encryption.
- **Privacy:** Respect user privacy and comply with relevant regulations.

By addressing these challenges and following the design principles outlined in this document, you can create a robust and scalable chat application similar to WhatsApp.